Applicant(s): Herbst, et al. Application No.: 10/696,624 Filed: 10/23/2003 Title: Apparatus to Offload and Accelerate Pico Code Processing Running in a Storage Processor Attorney Docket No.: 00121-002900000	Group Art Unit: 2187 Examiner: Brian R. Peugh

Amendments to the claims:

This listing of the claims will replace all prior versions, and listings, of claims in the application:

Listing of the claims:

1. (Currently Amended) A processor for offloading processing in a storage environment, comprising:

a zero bus turnaround ("ZBT") interface that interfaces said processor to a network processor configured to perform a storage function; [[and]]

a hash bucket memory having an array of bit-bucket pointers that each operate as a head pointer to a linked list of active semaphore structures, the hash bucket memory addressable by a hash address derived through application of a hash function to a semaphore value and wherein each semaphore structure accommodates a current thread that owns a semaphore and potentially one or more waiting threads in a queue waiting for release of the semaphore;

semaphore circuitry, coupled to said ZBT interface, that receives a signal from said network processor, and that controls [[a]] said semaphore associated with the corresponding predetermined linked list of active semaphore structures in the hash bucket memory wherein the received signal and said semaphore are related to ~~said signal for~~ locking and unlocking access to data in the storage environment; and[[.]]

an update engine, further included in the semaphore circuitry, that upon receipt of a received signal from said network processor relating to a thread on said network processor,

processes said semaphore related to said received signal and responds, as required, by sending a transmitted signal back to said network processor in association with said semaphore.

2. (Currently Amended) The processor of claim 1, wherein said update engine in the semaphore circuitry manages ~~a queue for~~ access to said semaphore by performing at least one command selected from a set of semaphore commands including: a semaphore request, a semaphore release and a thread exit.

3. (Currently Amended) The processor of claim 1 [[2]], wherein said update engine in the semaphore circuitry receives a second signal from said network processor and removes a request from said queue in response to said second signal when said network processor no longer desires said semaphore.

4. (Currently Amended) The processor of claim 1 [[2]], wherein said update engine in the semaphore circuitry refrains from sending to said network processor a second signal indicating said semaphore is unavailable, whereby said network processor continues to wait for said semaphore and said semaphore circuitry maintains ordered access to said queue.

5. (Original) The processor of claim 1, wherein said signal comprises one of a plurality of access requests for one of a plurality of semaphores, wherein said semaphore circuitry manages said

plurality of access requests in a plurality of queues, and wherein each of said plurality of queues corresponds to a respective one of said plurality of semaphores.

6. (Original) The processor of claim 1, wherein said semaphore circuitry comprises: a command queue that stores said signal received from said network processor.

7. (Currently Amended) The processor of claim 1, wherein said semaphore is [[a]] associated with a semaphore structure in the array of bit-bucket pointers in the hash bucket memory [[a hash array,]] and wherein said semaphore circuitry comprises:

a hash key generator that performs a hashing function on said signal for accessing said hash array.

8. (Currently amended) The processor of claim 1, wherein said update engine in the semaphore circuitry ~~comprises:an update engine that~~ receives a second signal from said network processor relating to a first process thread on said network processor, releases a lock on said semaphore related to said second signal, and sends a third signal to said network processor associating said semaphore with a second process thread on said network processor.

9. (Currently Amended) The processor of claim 1, wherein said update engine in the semaphore circuitry further comprises:

a semaphore queue manager that manages a queue of a plurality of semaphores.

10. (Currently Amended) The processor of claim 1, wherein said semaphore circuitry manages a queue for access to said semaphore, wherein said semaphore circuitry comprises:

a hash key generator that performs [[a]] the hashing function on said signal for accessing [[a hash array]] the hash bucket memory having an array of bit-bucket pointers, and that generates a hash key related to said signal, wherein said semaphore is [[a]] associated with a semaphore structure in said [[hash]] array of bit-bucket pointers, and wherein said signal relates to a first process thread on said network processor; and

[[an]] the update engine, coupled to said hash key generator, that receives said hash key, that releases a lock on said semaphore related to said hash key, and that sends a second signal to said network processor associating said semaphore with a second process thread on said network processor, wherein said first process thread and said second process thread are associated with said semaphore in said queue.

11. (Currently Amended) A method of controlling a processor for offloading processing in a storage environment, comprising the steps of:

receiving a signal from a network processor configured to perform a storage function through a zero bus turnaround ("ZBT") interface;

assigning a current thread as owner of a semaphore and potentially one or more waiting threads in a queue waiting for release of the semaphore to a semaphore structure in a linked list of active semaphore structures stored in a hash bucket memory addressable through application of a hash function to a semaphore value associated with the semaphore;

controlling [[a]] said semaphore associated with the linked list of active semaphore structures in the hash bucket memory wherein the received signal and the semaphore are related to ~~said signal for~~ locking and unlocking access to data in the storage environment; and [[.]]

processing the received signal from said network processor relating to a thread on said network processor and responding, as required, by sending a transmitted signal back to said network processor in association with said semaphore.

12. (Currently amended) The method of claim 11, wherein said step of controlling said semaphore further includes [[:]] managing ~~a queue for~~ access to said semaphore by performing at least one command selected from a set of commands including : a semaphore request, a semaphore release and a thread exit.

13. (Currently amended) The method of claim 11 [12], further comprising:

receiving a second signal from said network processor; and

removing a request from said queue in response to said second signal when said network processor no longer desires said semaphore.

14. (Currently amended) The method of claim 11 [12], further comprising:

refraining from sending to said network processor a second signal indicating said semaphore is unavailable, whereby said network processor continues to wait for said semaphore and said processor maintains ordered access to said queue.

15. (Currently amended) The method of claim 11 [12],, wherein said signal comprises one of a plurality of access requests for one of a plurality of semaphores, and wherein said step of controlling said semaphore includes:

managing said plurality of access requests in a plurality of queues, wherein each of said plurality of queues corresponds to a respective one of said plurality of semaphores.

16. (Currently amended) The method of claim 11 [12],, wherein said step of controlling said semaphore includes:

receiving a second signal from said network processor relating to a first process thread on said network processor;

releasing a lock in said semaphore related to said second signal; and

sending a third signal to said network processor associating said semaphore with a second process thread on said network processor.

17. (Currently amended) The method of claim 11, wherein said signal comprises one of a plurality of access requests for one of a plurality of semaphores, and wherein said step of controlling said semaphore includes:

managing said plurality of access requests in a plurality of queues, wherein each of said plurality of queues corresponds to a respective one of said plurality of semaphores;

receiving a second signal from said network processor relating to a first process thread on said network processor;

releasing a lock on said semaphore related to said second signal; and

sending a third signal to said network processor associating said semaphore with a second process thread for said network processor.

18-51. (Canceled)

52. (Canceled)

53. (Canceled)

54. (Canceled)

55. (Currently amended) A method comprising:

receiving a semaphore operation command from a network processor through an intra-system interface that identifies one of a plurality of semaphores in a linked list of active semaphore structures stored in a hash bucket memory addressable through application of a hash

function to a semaphore value associated with the semaphore; ~~, the command to identify one of a plurality of semaphores;~~

updating [[a data]] at least one of the active semaphore structures in the hash bucket memory ~~structure in a memory~~ according to the semaphore operation command received from the network processor through the intrasystem interface; and

returning a semaphore operation result to the network processor through an intra-system interface indicating an outcome of the semaphore operation command and responsive to the semaphore operation command received. ~~, the result to indicate an outcome of the command.~~

56. (Original) The method of claim 55, further comprising:

delaying the returning operation until after a second semaphore operation command from the network processor has been completed.

57. (Original) The method of claim 55 wherein the intra-system interface is a zero bus turnaround ("ZBT") interface.